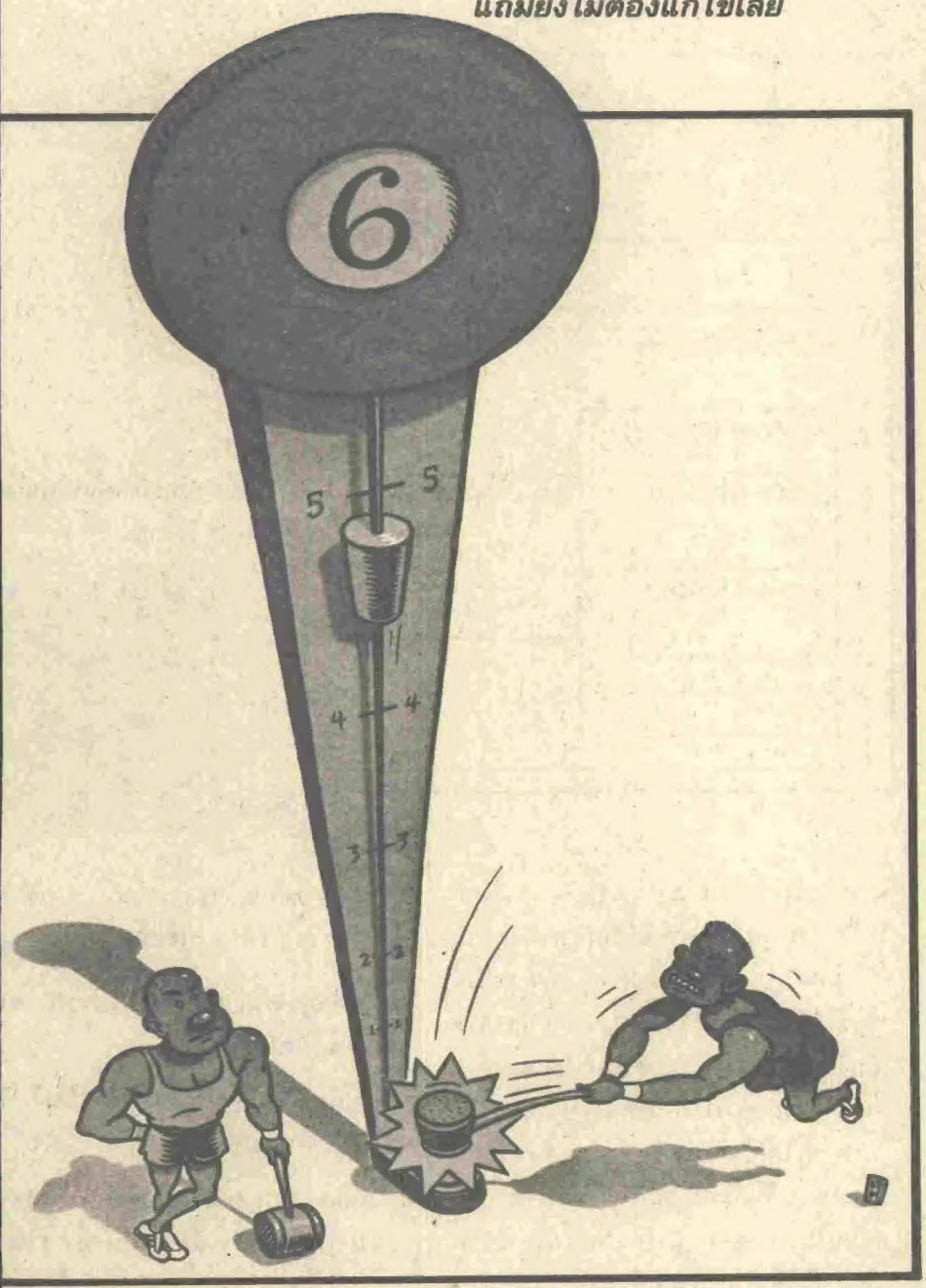
Windows 3.0
พัลลภ เฉลิมฉัตร

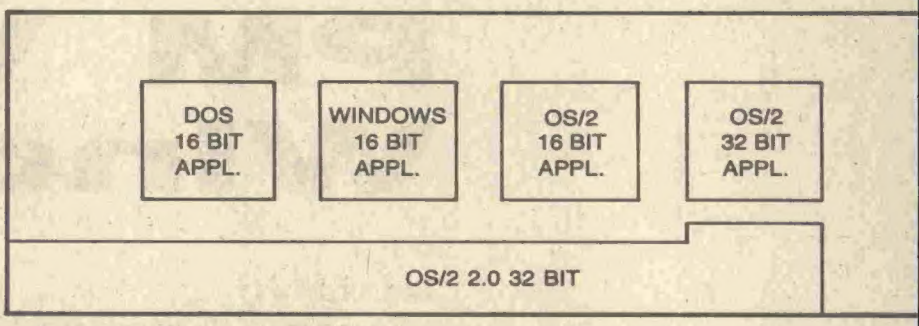
โปรแกรมมิ่งสไตล์ 2 เกลอ
PM VS Windows

เคล็ดลับที่น่ารู้
เมื่อ OS/2 รัน WINDOWS ได้
แถมยังไม่ต้องแก้ไขเลย



มื่อครั้งที่บริษัท ไอบีเอ็มแนะนำระบบปฏิบัติการโอเอสทูในเวอร์ชัน 1.1 มาพร้อมกับตัวกราฟิกยูสเซอร์อินเตอร์เฟสที่ชื่อ Presentation Manager (PM) หลาย ๆ คนกล่าวว่าพีเอ็มมีหน้าตาคล้ายคลึงกับวินโดว์ของบริษัทไมโครซอฟต์ในขณะนั้นมาก ซึ่งก็เป็นความจริงเพราะทั้งสองบริษัทยักษ์ใหญ่ได้ตกลงร่วมกันพัฒนาโอเอสทูและพีเอ็มจึงไม่เป็นที่แปลกใจถ้าบริษัทไมโครซอฟต์จะส่งทีมงานที่พัฒนาวินโดว์มาช่วยพัฒนาพีเอ็ม ความร่วมมือดังกล่าวดำเนินมาเรื่อย ๆ แต่ดูเหมือนจะไม่ได้รับความสนใจจากภายนอกจวบจนกระทั่งวินโดว์ 3.0 ออกสู่ตลาดและได้รับความสำเร็จอย่างมาก จึงเริ่มมีคนหันมาสนใจโอเอสทูและพีเอ็มมากขึ้นในทำนองว่าวินโดว์จะทำให้โอเอสทูไม่เป็นที่นิยมอีกนานพอสมควรเป็นผู้ยืดอายุของดอส ตลอดจนวินโดว์เป็นแผนทางการตลาดในการปูพื้นไปสู่โอเอสทูในอนาคต ถึงแม้ว่า DOS 5.0 จะออกมาแล้วในช่วงกลางเดือนมิถุนายนที่ผ่านมานี้พร้อมกับความสามารถที่เพิ่มเติมหลายอย่าง เช่น สามารถโหลดไดรเวอร์ขึ้นไปอยู่ในบริเวณ 640 กิโลไบต์ถึง 1 เมกะไบต์ ได้โดยใช้ devicehigh เพียงคำสั่งเดียว มีเอดิเตอร์ในตัว คำสั่งใหม่ ๆ เช่น undelete ฯลฯ แต่หลายคนยังคงพอใจที่จะรอคอยการมาของโอเอสทูเวอร์ชัน 2.0 ที่เป็นระบบจัดการสำหรับเครื่อง 386 จริง ๆ ไม่ใช่สำหรับเครื่อง-

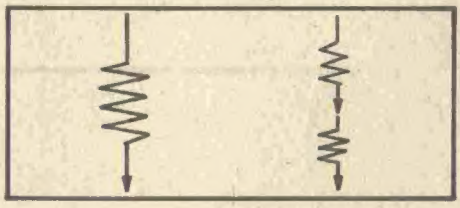
286 อย่างในโอเอสทูเวอร์ชันก่อน ๆ โดยมีโครงสร้างคร่าว ๆ สำหรับการรันโปรแกรมดังรูปที่ 1 ซึ่งระบบปฏิบัติการอันนี้อาจเข้ามาแทนดอสได้แน่นอนเสียที ผู้เขียนจึงใช้โปรแกรม Hello World สำหรับเปรียบเทียบวินโดว์และพีเอ็ม พร้อมทั้งแสดงโค้ดของโปรแกรมในไฟล์นามสกุล .C, .RC, .H และ .DEF



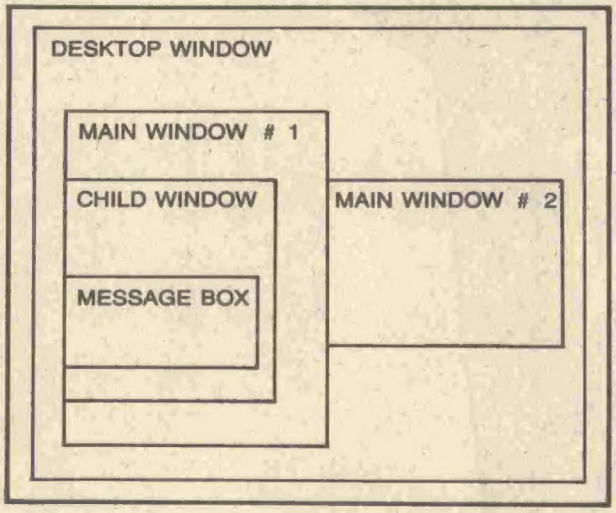
รูปที่ 1 แสดงชนิดของแอพพลิเคชันที่จะวิ่งอยู่บนโอเอสทู

มารู้จัก PM กันก่อน

เนื่องจากพีเอ็มถูกออกแบบให้วิ่งบนโอเอสทู ซึ่งเป็นระบบปฏิบัติการแบบมัลติทาสกิงที่แท้จริง ทำให้พีเอ็มพลอยมีประสิทธิภาพสูงขึ้นอันเนื่องจากความสามารถของระบบจัดการ ตามปกติในระบบมัลติทาสกิง (ระบบหลายงาน) จะมีโปรเซสทำงานอยู่ในระบบมากมาย ส่วนโอเอสทูจะทำให้แต่ละโปรเซสแตกออกเป็นส่วนย่อย ๆ เรียกว่าทรีด (thread) ย่อยแล้วทำงานขนานไปก็ได้เพื่อให้โปรแกรมใช้เวลาทำงานสั้นลงโดยเฉลี่ย ดูรูปที่ 2 ประกอบ นอกจากโปรแกรมเมอร์ที่เขียนโปรแกรมบนโอเอสทูสามารถออกแบบโปรแกรมเพื่อการแบ่งโปรเซส ออกเป็นหลาย ๆ ทรีดตามที่ต้องการแล้ว (ภายใต้ข้อจำกัดบางประการ) เช่น เมื่อใช้ซิสเท็มคอลล์แล้วโปรแกรมยังสามารถใช้บริการแบบอื่น ๆ เช่น พวก Interprocess Communication (pipe, semaphore ฯลฯ) ได้ด้วย โดยโอเอสทูออกแบบให้ซิสเท็มคอลล์ส่วนใหญ่เป็นภาษาซี แต่ออกแบบให้ง่ายต่อการเรียกใช้โดยภาษาอื่น เช่น โคบอล ฟอร์แทรน ถ้าเรามองพีเอ็มในแง่เทคนิคจะเห็นว่าเป็นเพียงยูสเซอร์อินเตอร์เฟสเชลล์เท่านั้น แต่ในทัศนะของโปรแกรมเมอร์ พีเอ็มดูเหมือนจะเป็นระบบปฏิบัติการเลยทีเดียว เหตุผลก็คือพีเอ็มมีแอพพลิเคชันโปรแกรมมิงอินเตอร์เฟส เครื่องมือของโปรแกรมเมอร์ในการพัฒนางานเป็นร้อยซึ่งสามารถทดแทนการใช้แอพพลิเคชันโปรแกรมมิงอินเตอร์เฟสของตัวโอเอสทูได้เป็นอย่างดี นอกจากนี้ยังอาจมองว่าพีเอ็มเป็นทูลบ็อกซ์ขนาดใหญ่ที่ช่วยในการพัฒนาโปรแกรมซึ่งมีหน้าตาและการใช้งานคล้ายคลึงกัน



รูปที่ 2 แสดงโปรเซสที่มีแค่ 1 ทรีด กับมี 2 ทรีดซึ่งแ
ทรีดจะทำงานไปพร้อม ๆ กัน



รูปที่ 3 แสดงลำดับชั้นของวินโด
ในระบบ

Window Hierarchy บนพีเอ็ที่แสดงในรูปที่ 3 ประกอบด้วย

- Desktop Window คืออีกชื่อหนึ่งขดิสเพลย์สกรีน
- Main Window คือชื่อของวินโดว์เปิดเป็นแอพพลิเคชัน
- Child Window คือชื่อวินโดว์ย่ของแอพพลิเคชันที่เปิดเพื่องานงานหนึ่ง

Dialog Box คือชื่อของวินโดว์ย่อยที่ถูกเปิดขึ้นมาเพื่อการติดต่อกับผู้ใช้

ฤษฎีการเขียนโปรแกรมสำหรับพีเอ็ม

แอพพลิเคชันของพีเอ็มมีลักษณะล้ายกับแอพพลิเคชันบนวินโดว์ คือมีำดับขั้นตอนของการเขียนโปรแกรมค่อนข้างแน่ชัด (แต่เราไม่จำเป็นต้องำตามทุกกรณีก็ได้ มีโอกาสผู้เขียนจะสดงโปรแกรมบนวินโดว์ที่ไม่เกิน 100 รรทัด)

สำหรับขั้นตอนของพีเอ็มมีดังนี้

1. Initial PM
2. สร้าง massage queue สำหรับอพพลิเคชัน
3. Register Window Class
4. Create Window
5. Execute Loop
6. Exit from Loop แล้วทำการestroy window, destroy queue และกเลิก window environment สำหรับอพพลิเคชันนั้น

สายสัมพันธ์ SAA กับพีเอ็ม

พีเอ็มมีความสัมพันธ์ใกล้ชิดอย่างมากกับ SAA (System Application Architecture) ของไอบีเอ็ม โดยออกแบบมาเพื่อรองรับการโยกย้ายแอพพลิเคชันและข้อมูลไปมาระหว่างเครื่องคอมพิวเตอร์ตระกูลต่าง ๆ ของไอบีเอ็ม นับตั้งแต่เครื่องไมโครคอมพิวเตอร์เครื่องในตระกูล S/3x ไปจนถึง S/370 ซิสเต็มแอพพลิเคชันอาร์คิเทกเจอร์ได้กำหนดชุดของการอินเตอร์เฟสและโปรโตคอลไว้มากมายพอสมควร ส่วนที่มีความสัมพันธ์กับพีเอ็มอย่างมากก็คือ

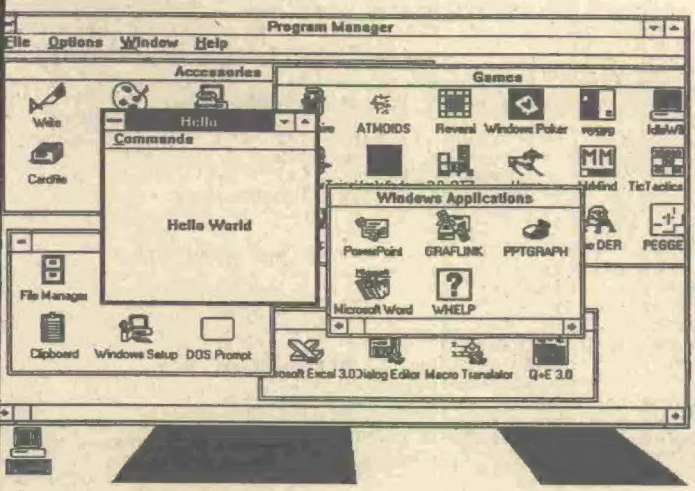
- คอมมอนยูสเซอร์แอคเซสเป็นมาตรฐานการออกแบบหน้าตาของยูสเซอร์อินเตอร์เฟสในการทำงานและติดต่อกับผู้ใช้ เช่น เมนูรายการ "About..." ควรอยู่ภายใต้เมนูหัวข้อซ้ายสุดหรือขวาสุด

- คอมมอนโปรแกรมมิงอินเตอร์เฟสกำหนดชุดของอินเตอร์เฟสที่แอพพลิเคชัน

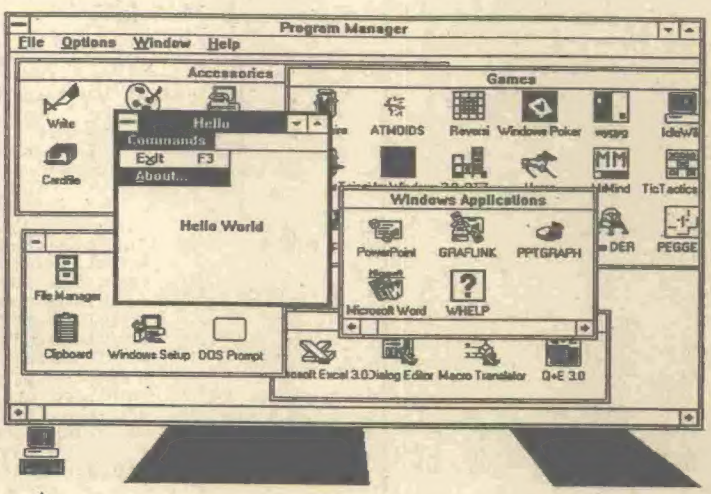
ทดสอบการทำงานด้วยโปรแกรมตัวอย่าง

เมื่อโปรแกรม Hellowin และ Hellopm รันอยู่ในระบบจัดการชนิดใดชนิดหนึ่ง เมื่อเราเปิดวินโดว์จะมีข้อความว่า "Hello World" ปรากฏบริเวณส่วนกลางของไคลเอ็นต์แอเรีย ดังรูปที่ 4 เมื่อเลือก (โดยใช้เมาส์หรือ Alt-C) จะเปิดรายการในเมนูพร้อมกับแสดงรายการเมนู 2 ประเภทคือ Exit และ About... ซึ่งการเลือกรายการอาจทำได้โดยการใช้เมาส์หรือกดปุ่มตัวอักษรที่มีขีดเส้นใต้ก็ได้ แต่ในรายการ Exit จะมีฮอตคีย์ F3 ให้ด้วย ดังแสดงในรูปที่ 5 เมื่อเลือกรายการ About... โปรแกรมจะเปิดแมสเสจบ็อกซ์ พร้อมทั้งแสดงข้อความและรอให้ผู้ใช้กดปุ่ม OK เพื่อกลับไปทำงานต่อ (รูปที่ 6) ถ้ามีการเปลี่ยนแปลงขนาดของวินโดว์หรือย้ายตำแหน่ง โปรแกรมจะคำนวณหาขนาดของไคลเอ็นต์แอเรียใหม่ ตลอดจนแสดงข้อความตรงบริเวณส่วนกลางของไคลเอ็นต์ (รูปที่ 7)

ต่อไปจะเป็นการเปรียบเทียบการ

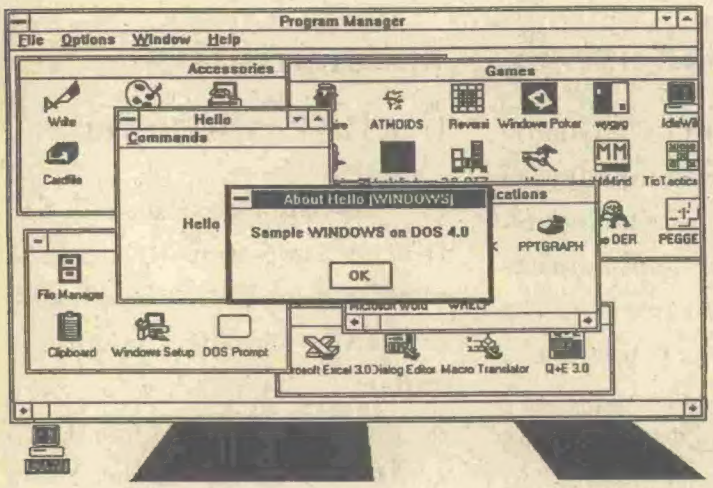


ปที่ 4

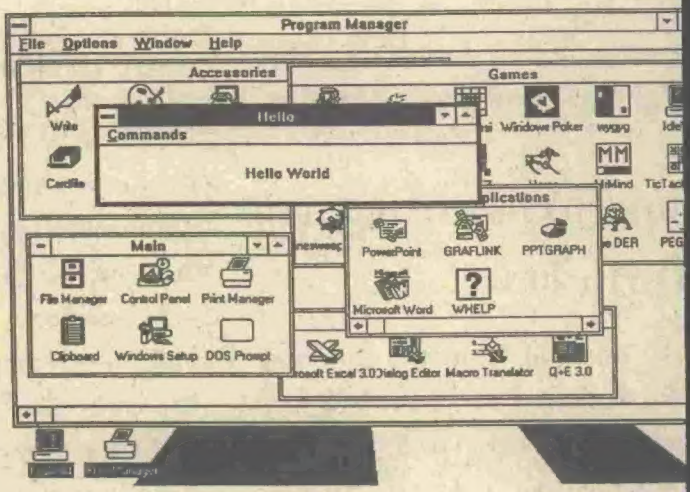


รูปที่ 5

Windows 3.0
โปรแกรมมิงสไตล์ 2 เกลอ PM VS Windows



รูปที่ 6



รูปที่ 7

ทำงานโดยพิจารณาเฉพาะจุดสำคัญของโปรแกรม Hellowin.C และ Hellopm.C ดังรูปที่ 8 และ 9 ตามลำดับ

จุดที่ 1 เป็นการ include header file ซึ่งทั้งในพีเอ็มและวินโดว์ต่างก็มีแฟ้มหัวเรื่องของตนเองคือ os2.h และ windows.h ตามลำดับ แต่จะไม่เหมือนกันเพราะใน PM os2.h ไม่ได้กำหนดหรือประกาศตรงอย่างใน windows.

รูปที่ 8 โปรแกรม Hellowin.C

```
/***

    Program:        Hello Windows
    By:             Mr.  Punlop Chalermchat
    Date:           21 June 1991
    Note:           OS   :   DOS  version 4.0

***/
#include <windows.h>                    /* required for all Windows applications */
#include "hellowin.h"

/***------------------------------------------------------------------------***
    Function    Prototype
***------------------------------------------------------------------------***/
long FAR PASCAL WndProc(HWND,WORD,WORD,LONG);
int PASCAL WinMain(HANDLE,HANDLE,LPSTR,int);

HWND     hwnd;                          /* Window handle */
static char szAppName[] = "HelloWin";   /* String for window class */
HANDLE   hAccel;                        /* Accelerator table handle */
MSG      msg;                           /* Message from message queue */
WNDCLASS wndclass;                      /* Window class */
RECT     rect;                          /* Window creation control flags */


/***-------------------------Start of main procedure ----------------------***
int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)

HANDLE hInstance;                   /* current instance  */
HANDLE hPrevInstance;               /* previous instance */
LPSTR lpCmdLine;                    /* command line      */
int nCmdShow;                       /* show-window type (open/icon) */
{

    if (!hPrevInstance)  {          /* Other instances of app running? */
        wndclass.style = CS_HREDRAW | CS_VREDRAW;
        wndclass.lpfnWndProc   = WndProc;
        wndclass.cbClsExtra    = 0;
        wndclass.cbWndExtra    = 0;
        wndclass.hInstance     = hInstance;
        wndclass.hIcon         = LoadIcon(NULL,IDI_APPLICATION);
        wndclass.hCursor       = LoadCursor(NULL,IDC_ARROW);
        wndclass.hbrBackground= GetStockObject(WHITE_BRUSH);
        wndclass.lpszMenuName = szAppName;      /* Set menu name in .RC  */
        wndclass.lpszClassName= szAppName;      /* Set window class Name */

        RegisterClass(&wndclass);               /* Register window class */
```

จุดที่ 1

จุดที่ 2

รูปที่ 8 (ต่อ)

```
จุดที่ 4     hwnd = CreateWindow( szAppName,              /* Create window */
                                  "Hello",                /* Window title text */
                                  WS_OVERLAPPEDWINDOW,
                                  100,100,200,200,
                                  NULL,
                                  NULL,
                                  hInstance,
                                  NULL );

             ShowWindow(hwnd,nCmdShow);
             UpdateWindow(hwnd);
จุดที่ 5     hAccel = LoadAccelerators(hInstance,szAppName);

          /***-------------------------------------------------------------------------

             Get and dispatch messages from the application message queue
             until WinGetMsg returns FALSE, indicating a WM_QUIT message.
          -------------------------------------------------------------------------***/
จุดที่ 6     while (GetMessage(&msg,     /* message structure  */
                NULL,                   /* handle of window receiving the message */
                0,                      /* lowest message to examine */
                0))                     /* highest message to examine */
             {
             if (!TranslateAccelerator(hwnd,hAccel,&msg)  )
             {
                TranslateMessage(&msg);    /* Translates virtual key codes  */
                DispatchMessage(&msg);     /* Dispatches message to window  */
             }
             }
จุดที่ 7     return (msg.wParam);          /* Returns the value from PostQuitMessage */

          /***------------------  End of main procedure  ------------------------***/

          /***------------------  Start of window procedure ---------------------***/
          long FAR PASCAL WndProc(hwnd, message, wParam, lParam)
          HWND hwnd;                    /* window handle          */
          unsigned message;             /* type of message        */
          WORD wParam;                  /* additional information */
          LONG lParam;                  /* additional information */
          {
          HDC hdc;                      /* Handle device context  */
          PAINTSTRUCT  ps;              /* paint structure        */
          RECT         rect;            /* Rectangular            */

             switch (message) {
จุดที่ 9     case WM_COMMAND:
                   switch(wParam)    {
                case IDM_EXITPROG:
                        PostQuitMessage(0);
                        break;
                case IDM_ABOUT:
                    MessageBox(hwnd,
                                "Sample WINDOWS on DOS 4.0",
                                "About Hello (WINDOWS)",
                                MB_OK  );
                    break;
                default:
                    return DefWindowProc(hwnd,message,wParam,lParam);
             }
จุดที่ 10    case WM_PAINT:
                   hdc = BeginPaint(hwnd,&ps);   /* Create a presentation space */
                   GetClientRect(hwnd,&rect);
                   DrawText(hdc,"Hello World",-1,&rect,
                         DT_SINGLELINE | DT_CENTER | DT_VCENTER );
                   EndPaint(hwnd,&ps);           /* Drawing is complete */
                   return 0;

จุดที่ 11    case WM_DESTROY:              /* message: window being destroyed */
                PostQuitMessage(0);
                break;

             default:                  /* Passes it on if unproccessed    */
                return (DefWindowProc(hwnd, message, wParam, lParam));


             return (NULL);
          }


          /***-------------------  End of window procedure  ---------------------***/
```

รูปที่ 9 โปรแกรม Hellopm.C

```
/***

    Program:        Hello Presentation Manager
    By:             Mr.  Punlop Chalermchat
    Date:           21 June 1991
    Note:           OS :  Operating System /2     version 1.3

***/

#define INCL_PM      /* for include appropriate  header file */

#include <os2.h>                        /* OS/2  header file  */
#include <string.h>                     /* C/2 string functions */
#include "hellopm.h"                    /* program header file  */


/***--------------------------------------------------------------------***
    Function    Prototype
***--------------------------------------------------------------------***/

MRESULT EXPENTRY WndProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2 );
void cdecl main( void );


#define STRINGLENGTH 20                 /* Length of string */
HMQ  hmq;                               /* Message queue handle          */
HWND hwndClient;                        /* Client area window handle     */
HWND hwndFrame;                         /* Frame window handle           */
QMSG qmsg;                              /* Message from message queue    */
ULONG flCreate;                         /* Window creation control flags */

HAB  hab;                               /* PM anchor block handle        */
CHAR szHello[STRINGLENGTH], szWorld[STRINGLENGTH],
                            szString[STRINGLENGTH];

                                        /* String parameters, set in     */
                                       /* the processing of WM_CREATE, */

/***-------------------------Start of main procedure ------------------------***/
void cdecl main(  )
{
    hab = WinInitialize( NULL );        /* Initialize PM    */
    hmq = WinCreateMsgQueue( hab, 0 );  /* Create a message queue    */

    WinRegisterClass(                   /* Register window class    */
                hab,                    /* Anchor block handle       */
                "HelloPM"               /* Window class name        */
                WndProc,                /* Address of window procedure  */
                CS_SIZEREDRAW,          /* Class style                */
                0                       /* No extra window words    */
     );

     /* Set frame control flags */
                flCreate = FCF_SYSMENU    |
                           FCF_TITLEBAR   |
                           FCF_SIZEBORDER |
                           FCF_MINMAX     |
                           FCF_MENU       |
                           FCF_ACCELTABLE ;

  hwndFrame = WinCreateStdWindow(
                HWND_DESKTOP,           /* Desktop window is parent  */
                0L,                     /* No frame styles           */
                &flCreate,              /* Frame control flag        */
                "HelloPM",              /* Client window class name  */
                "Hello",                /* Window title text         */
                0L,                     /* No special class style    */
                NULL,                   /* Resource is in .EXE file  */
                ID_WINDOW,              /* Frame window identifier   */
                &hwndClient             /* Client window handle      */
                );

   WinSetWindowPos( hwndFrame,          /* Shows and activates frame */
                    HWND_TOP,           /* window at position 100, 100, */
                    100, 100, 200, 200, /* and size 200, 200.          */
                    SWP_SIZE | SWP_MOVE | SWP_ACTIVATE | SWP_SHOW
                  );


  /***--------------------------------------------------------------------
      Get and dispatch messages from the application message queue
      until WinGetMsg returns FALSE, indicating a WM_QUIT message.
  ----------------------------------------------------------------------***/
    while( WinGetMsg( hab, &qmsg, NULL, 0, 0 ) )
      WinDispatchMsg( hab, &qmsg );
```

รูปที่ 9 (ต่อ)

```
จุดที่ 7   WinDestroyWindow( hwndFrame );          /* Tidy up...    */
           WinDestroyMsgQueue( hmq );              /* and           */
           WinTerminate( hab );                    /* terminate the application  */
         }
         /***------------------  End of main procedure  ------------------------***/

         /***------------------  Start of window procedure ---------------------***/

         MRESULT EXPENTRY WndProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2 )
         {
           USHORT command;              /* WM_COMMAND command value      */
           HPS    hps;                  /* Presentation Space handle     */
           RECTL  rc;                   /* Rectangle coordinates         */
           POINTL pt;                   /* String screen coordinates     */
           RECTL  rect ;                /* Rectangular                   */
           switch( msg )
           {
จุดที่ 8     case WM_CREATE:
               /***-------------------------------------------------------------------------
                        Window initialization is performed here in WM_CREATE processing
                         WinLoadString loads strings from the resource file.
               -------------------------------------------------------------------------***/
               WinLoadString( hab, NULL, IDS_HELLO, STRINGLENGTH, szHello );
               WinLoadString( hab, NULL, IDS_WORLD,     STRINGLENGTH, szWorld );
               strcpy( szString, szHello );      /* Copy text "Hello World" into szString
               strcat( szString, szWorld );
                /***
              strcpy(szString,"Hello World");
              ^-------- You can replace those statements by this one.
               ***/
               break;

จุดที่ 9        case WM_COMMAND:
                     command = SHORT1FROMMP(mp1);   /* Extract the command value    */
                switch (command)   {
                     case IDM_EXITPROG:
                         WinPostMsg( hwnd, WM_CLOSE, 0L, 0L );
                         break;

                     case IDM_ABOUT:
                         WinMessageBox(HWND_DESKTOP,hwndClient,
                        " Sample PM on OS/2 version 1.3",
                          "About Hello (PM)",
                                NULL,MB_OK);
                               break;
                   default:
                  return WinDefWindowProc( hwnd, msg, mp1, mp2 );
                  }
               break;

จุดที่ 10   case WM_PAINT:
              WinQueryWindowRect(hwnd,&rect);   /* Create a presentation space  */
              hps = WinBeginPaint( hwnd, NULL, &rc );
              WinDrawText(hps,-1,szString,&rect,CLR_NEUTRAL,
                   CLR_BACKGROUND,DT_CENTER | DT_VCENTER |
                                  DT_ERASERECT );
              WinEndPaint( hps );               /* Drawing is complete   */
              break;

จุดที่ 11   case WM_CLOSE:
              WinPostMsg( hwnd, WM_QUIT, 0L, 0L );  /* Cause termination  */
              break;
            default:
             return WinDefWindowProc( hwnd, msg, mp1, mp2 );
           }
           return FALSE;
         }

         /***------------------  End of window procedure  ----------------------***/
```

รูปที่ 10 โปรแกรม Hellowin. RC

```
#include <windows.h>
#include "hellowin.h"

HELLOWIN        MENU
{
        POPUP   "&Commands"       {
        MENUITEM "E&xit\tF3",IDM_EXITPROG
        MENUITEM "&About...",IDM_ABOUT
        }
}

HELLOWIN   ACCELERATORS
{
        VK_F3,IDM_EXITPROG,VIRTKEY
}
```

```
#include <os2.h>
#include "hellopm.h"

MENU    ID_WINDOW PRELOAD
BEGIN
   SUBMENU "~Commands",IDM_OPTIONS
   BEGIN
    MENUITEM "E~xit\tF3", IDM_EXITPROG, MIS_TEXT
    MENUITEM "~About...",     IDM_ABOUT,MIS_TEXT
   END
END

ACCELTABLE ID_WINDOW PRELOAD
BEGIN
    VK_F3, IDM_EXITPROG, VIRTUALKEY
END

STRINGTABLE PRELOAD
BEGIN
 IDS_HELLO, "Hello"
 IDS_WORLD,"  World"
END
```

รูปที่ 11 โปรแกรม Hellopm.C

หากเป็นเพียงการตรวจสอบค่าที่กำหนดไว้ตอนต้นโปรแกรม (โปรดสังเกตประโยค #define INCL__PM) เพื่อให้โปรแกรมสามารถเลือก include เฉพาะแฟ้มหัวเรื่องที่จำเป็นต้องใช้เข้ามา ดังรูปที่ 16

จุดที่ 2 เป็นการกำหนดค่าเริ่มต้น (initial) ให้ระบบโดย

วินโดว์ : จะตรวจสอบว่าในระบบเคยมีบางส่วน ของแอพพลิเคชันนี้รัน

```
#define  IDM_EXITPROG     1
#define  IDM_ABOUT        2
```

รูปที่ 12 Hellowin.H

```
#define ID_WINDOW      256
#define IDM_OPTIONS    257
#define IDM_EXITPROG   258
#define IDM_ABOUT      259
#define IDS_HELLO      262
#define IDS_WORLD      263
```

รูปที่ 13 Hellopm.H

```
NAME            HELLOWIN
DESCRIPTION     'Demo Windows Programming Style '
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       8192
EXPORTS         WndProc @1
```

รูปที่ 14 Hellowin.DEF

```
NAME            hellopm WINDOWAPI

DESCRIPTION     'Demo PM Programming Style'
PROTMODE
HEAPSIZE        4096
STACKSIZE       8192
EXPORTS          WndProc
```

รูปที่ 15 Hellopm.DEF

รูปที่ 16 OS2.H

```
/****************************** Module Header ******************************\
*
* Module Name: OS2.H
*
* This is the top level include file that includes all the files necessary
* for writing an OS/2 application.
*
* Copyright (c) International Business Machines Corporation 1981, 1988, 1989
* Copyright (c) Microsoft Corporation 1981, 1988, 1989
*
\*************************************************************************/

#define OS2_INCLUDED

/* Common definitions */

#include <os2def.h>

/* OS/2 Base Include File */

#include <bse.h>

/* OS/2 Presentation Manager Include File */

#include <pm.h>
```

อยู่ก่อนหรือไม่ จะได้ทำการ share code แต่ใช้ข้อมูลแยกจากกัน ถ้าไม่มีก็จะกำหนดคุณสมบัติของวินโดว์ที่ต้องการเปิด

พีเอ็ม : ทำการสร้างแอนชอร์-บล็อกซึ่งเป็นพื้นที่ในหน่วยความจำที่ DDL ทั้งหลายของระบบนำไปใช้ในการเก็บข้อมูลเกี่ยวกับแอพพลิเคชันตัวนั้น สำหรับรายละเอียดที่เก็บไว้ในแอนชอร์บล็อกจะไม่เป็นที่เปิดเผย ยกเว้นผู้ที่พัฒนาพีเอ็มเท่านั้น หลังจากสร้างแอนชอร์บล็อกแล้ว ขั้นต่อไปจะสร้างแมสเสจคิวให้แก่แต่ละแอพพลิเคชันหรือแมสเสจคิวของแต่ละทรีด โดยหนึ่งแอพพลิเคชันมีได้หลายทรีด แต่ละทรีดมีคิวของมันเอง แต่ปกติเรากำหนดให้มีแมสเสจคิวเพียงหนึ่งคิวเท่านั้นสำหรับแต่ละแอพพลิเคชัน

จุดที่ 3 พีเอ็มจะทำการตั้งค่าเฟรมคอนโทรลของวินโดว์ ว่าจะแสดงหน้าตาอย่างไรบ้าง เช่น มีไตเติลบาร์ ปุ่ม Min/Max ฯลฯ

จุดที่ 4 วินโดว์จะทำการสร้างวินโดว์แต่ยังไม่แสดงบนจอภาพจริง โดยรูปแบบหน้าต่าง ของวินโดว์ที่จะสร้างคือ WS_OVERLAPPED-WINDOW ซึ่งเป็นสไตล์ที่มี ไตเติลบาร์ซิสเต็มเมนูบาร์ ไซซ์ซิ่งบอร์เดอร์ ฯลฯ สำหรับการเลือกสไตล์ต่าง ๆ สามารถกำหนดได้จากค่าแฟล็กที่กำหนดไว้ใน windows.h ในการสร้างวินโดว์เราสามารถระบุได้ว่าจะให้ปรากฏ ณ จุดใด ซึ่งในที่นี้กำหนดไว้ที่ (100,100) ขนาด 200 ทั้งแกน x และy แต่ทั้งนี้เราอาจใช้ค่า CW_USEDEFAULT แทนก็ได้ แล้วใช้ฟังก์ชัน SetWindow Pos ช่วยในกรณีที่เราต้องย้ายตำแหน่งไปจุดที่ต้องการในภายหลัง การแสดงวินโดว์ให้ปรากฏเราใช้คำสั่ง Show Window โดยกำหนดค่าในพารามิเตอร์ว่าจะให้แสดงอย่างไร (เป็นวินโดว์หรือเป็นไอคอน) แล้วใช้คำสั่ง Update Window ซึ่งจะส่ง WM_PAINT แมสเสจไปให้ WndProc ทำให้ผู้ใช้สามารถวาดข้อมูลในไคลเอ็นต์แอเรียใหม่

ในพีเอ็ม : คำสั่ง WinCreateStd Window จะสร้างวินโดว์ตามที่เรากำหนดทุกอย่าง นอกจากนี้ยังสามารถใช้ฟังก์ชัน WinCreateWindow แทนก็ได้ แต่ต่างกันตรงที่ WinCreateWindow จะสร้างแค่หนึ่งวินโดว์ ในขณะที่ Win-CreateStdWindow จะสร้าง 2 วินโดว์ คือเฟรมวินโดว์และไคลเอ็นต์วินโดว์ หลังจากสร้างวินโดว์แล้วจะต้องเซตตำแหน่งโดยใช้ WinSetWindowPos

จุดที่ 5 วินโดว์จะโหลดค่าฮอตคีย์จากแฟ้มต้นฉบับมาใช้ผ่านคำสั่ง Load accelerator ในขณะที่ในพีเอ็มกำหนดไว้ด้วยแฟล็ก FCF_ACCELTABLE

จุดที่ 6 วินโดว์จะรับแมสเสจมาตีความโดย Translate accelerator ก่อนส่งต่อไปยังวินโดว์โพรซีเดอร์ที่

เหมาะสมต่อไป (ถ้าเราไม่ใช้ประโยค Translate accelerator ถึงแม้จะ Load accelerator มาแล้วก็ตามฮอตคีย์ย่อมไม่มีประโยชน์)

โปรแกรมพีเอ็มจะรับแมสเสจมาเพื่อตีความก่อนส่งให้วินโดว์โพรซีเดอร์ที่เหมาะสม

จุดที่ 7 วินโดว์จะคืนค่าจาก PostQuitMessage เก็บไว้ในพารามิเตอร์ขนาด word (2 ไบต์)

พีเอ็มจะยกเลิกวินโดว์ (WinDestroyWindow), ยกเลิกคิว (WinDestroyMsgQueue) และยกเลิกแอนชอร์บล็อก (WinTerminate)

จุดที่ 8 ในพีเอ็มเมื่อเราสร้างวินโดว์ขึ้น ตัววินโดว์จะรับข้อความจากระบบคือ WM_CREATE ซึ่งในโปรแกรมใช้ในการโหลดค่าสตริงจากตารางในแฟ้มต้นฉบับ เราอาจใช้ฟังก์ชัน strcpy ก็ได้ แต่ตัวอย่างที่แสดงเป็นการใช้ข้อมูลจากแฟ้มต้นฉบับ ซึ่งในวินโดว์สามารถกำหนดตารางสตริงได้เช่นกัน

จุดที่ 9 เมื่อเราเลือกเมนูในวินโดว์ด้วยการเลือกรายการเมนู ในหัวข้อตัวโปรแกรมจะดักรับแมสเสจจาก WM_COMMAND message ซึ่งมีค่าหมายเลขประจำตัวของรายเมนูที่คัดเลือกแล้วส่งให้เวิร์ดพารามิเตอร์ โดยในกรณีที่เราเลือกหัวข้อ "Exit" ก็จะส่ง WM_QUIT ไปเก็บไว้ในแมสเสจคิว ส่วนในกรณีที่เลือกหัวข้อ "About..." ก็จะเปิดแมสเสจบอกซ์แสดงข้อความ

ในพีเอ็มค่าของหัวข้อที่เลือกอยู่ในบางส่วนของ mpl ดังนั้นจึงต้องใช้คำสั่งดึงค่าออกมาอีกทีก่อนที่จะตรวจเลือกรายการเมนู

จุดที่ 10 วินโดว์จะคำนวณขนาดของไคลเอ็นต์แอเรียและแสดงข้อความตรงบริเวณใจกลางของไคลเอ็นแอเรีย ส่วนพีเอ็มทำแบบเดียวกัน หากแต่ในที่นี้ใช้ตัวแปรสตริงที่โหลดค่าจากแฟ้มต้นฉบับ

จุดที่ 11 เมื่อเราเลือกหัวข้อ Close ในเมนูหลักของวินโดว์ โปรแกรมจะส่ง WM_SYSCOMMAND ไปให้วินโดว์โพรซีเดอร์ (WndProc) และส่งต่อไปให้ DefWindowProc ซึ่งจะส่ง WM_CLOSE ออกมาให้วินโดว์โพรซีเดอร์และส่งต่อไปให้ DefWindowProc ซึ่งจะส่ง WM_DESTROY มาจัดแมสเสจคิวซึ่งในโปรแกรมก็จะทำการตอบสนองต่อ WM_DESTROY ด้วยการใช้คำสั่ง PostQuitMessage อันจะเป็นการส่ง WM_QUIT ไปใส่ในแมสเสจคิว อันเป็นผลให้โปรแกรมหยุดการทำงาน (หลุดออกจากลูป) แต่พีเอ็มจะทำงานแบบเดียวกัน ในที่นี้ใช้การดักที่ WM_CLOSE แทน

เราจะสังเกตได้ว่า WM_CLOSE จะมาก่อน WM_DESTROY เสมอ ทั้งนี้เพราะเราสามารถที่จะทำงานอย่างหนึ่งอย่างใดเมื่อได้รับ WM_CLOSE เช่น เตือนว่าไฟล์ยังไม่ได้เซฟ เป็นต้น ในขณะที่ถ้า WM_DESTROY จะมีการปิดวินโดว์โดยหลีกเลี่ยงไม่ได้ เราควรดักที่ WM_CLOSE แต่ที่น่ากลัวที่สุดก็คือ WM_QUIT ซึ่งทำให้โปรแกรมหลุดออกจากลูป หมายถึงการปิดทั้งแอพพลิเคชันไม่ใช่แค่วินโดว์ใดวินโดว์หนึ่งเหมือน WM_DESTROY (ข้อควรสังเกต รูปแบบการออกจากโปรแกรมโดยปกติของวินโดว์อยู่ที่ WM_DESTROY ในขณะที่โปรแกรมพีเอ็มจะอยู่ที่ WM_CLOSE)

ข้อสังเกตอีกประการหนึ่งก็คือ การกำหนดค่าหมายเลขประจำตัวในแฟ้มต้นฉบับของพีเอ็มจะกำหนดค่าตั้งแต่ 256 ขึ้นไป (ค่า 0-255 เก็บไว้ให้ระบบใช้)

สรุป

ผู้เขียนได้เสนอความแตกต่างบางจุดของโปรแกรมบนวินโดว์กับพีเอ็มให้ผู้อ่านได้พิจารณา เราจะเห็นว่าทั้งสองโปรแกรมมีความคล้ายคลึงกันมาก ดังนั้นจึงไม่เป็นที่แปลกใจ ถ้าโอเอสทู 2.0 จะสามารถรันแอพพลิเคชันในวินโดว์ได้โดยไม่ต้องแก้ไขเลย อีกทั้งผู้ที่เคยเขียนโปรแกรมบนวินโดว์ที่จะสามารถขยับไปเขียนโอเอสทูได้อย่างไม่ยากเย็น